在Linux运行期间升级Linux系统（Uboot+kernel+Rootfs）

版本：v1.2

Crifan Li

摘要

本文主要介绍了如何在嵌入式Linux系统运行的时候，进行升级整个Linux系统，包括uboot，kernel和rootfs。以及简介Linux中的已有的通用的Nor Flash驱动m25p80，和简介mtd util以及相关工具mtdinfo,flash_erase,flash_eraseall,nanddump,nandwrite等的基本用法。

本文提供多种格式供：

在线阅读	HTML [1]	HTMLs [2]	PDF [3]	CHM [4]	TXT [5]	RTF [6]	WEBHELP [7]
下载（7zip压缩包）	HTML [8]	HTMLs [9]	PDF [10]	CHM [11]	TXT [12]	RTF [13]	WEBHELP [14]

HTML版本的在线地址为：

http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/html/runtime_upgrade_linux.html

有任何意见，建议，提交bug等，都欢迎去讨论组发帖讨论：

http://www.crifan.com/bbs/categories/runtime_upgrade_linux/

修订历史		
修订 1.0	2011-05-03	crl
1. 介绍了如何实现在线升级linux系统，即uboot，kernel，rootfs, 以及相关的前提知识和准备工作		
修订 1.2	2012-11-17	crl
1. 通过Docbook发布 2. 添加了各章节的id		

[1] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/html/runtime_upgrade_linux.html
[2] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/htmls/index.html
[3] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/pdf/runtime_upgrade_linux.pdf
[4] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/chm/runtime_upgrade_linux.chm
[5] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/txt/runtime_upgrade_linux.txt
[6] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/rtf/runtime_upgrade_linux.rtf
[7] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/webhelp/index.html
[8] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/html/runtime_upgrade_linux.html.7z
[9] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/htmls/index.html.7z
[10] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/pdf/runtime_upgrade_linux.pdf.7z
[11] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/chm/runtime_upgrade_linux.chm.7z
[12] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/txt/runtime_upgrade_linux.txt.7z
[13] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/rtf/runtime_upgrade_linux.rtf.7z
[14] http://www.crifan.com/files/doc/docbook/runtime_upgrade_linux/release/webhelp/runtime_upgrade_linux.webhelp.7z

3. 建议挂载文件到ramdisk中

在Linux运行期间升级Linux系统（Uboot+kernel+Rootfs）：

Crifan Li

版本：v1.2

出版日期 2012-11-17

[15] http://www.crifan.com/files/doc/docbook/soft_dev_basic/release/html/soft_dev_basic.html#cc_by_nc

目录

插图清单

表格清单

正文之前

1. 此文目的

目前嵌入式Linux系统的升级，即升级uboot，kernel，rootfs等，的传统的方式，都是用烧写工具去烧写，相对来说，显得很繁琐和效率比较低，而利用mtd工具的方式去升级系统，相对比较方便。

此文主要就是介绍，在嵌入式Linux系统下，已经实现了nand和（或）nor flash驱动后，如何利用mtd工具，进行实时（runtime）/在线（online）的情况下，升级Linux系统。

2. 一点说明

1. 本文所写内容，主要是之前的一些相关的工作总结，如果内容有误，请及时告知：admin (at) crifan.com

   其他技术问题的探讨，任何的问题，意见，建议等，都欢迎邮件交流。

2. 另外，如果需要的mtd-utils-1.3.1的源码的话，也可以发邮件索取。

3. 有此文相关的有两个附件：

   - compiled_mtd-utils_arm.7z [1]

     已经编译好了的arm平台的，包含了u32和u64版本的，本文所用到的那4个mtd 的工具，即flash_erase，flash_eraseall，nanddump，nandwrite。

   - mtd-utils-1.3.1_support_u32u64.7z [2]

     我之前所用的mtd util的源码。你如果是其他平台的，那么用此源码，可以自己编译出对应的mtd的一系列的工具。关于如何编译，请参考Readme文件。

[1] http://www.crifan.com/files/tool/embedded/linux/mtd/mtd_utils/compiled_mtd-utils_arm.7z
[2] http://www.crifan.com/files/tool/embedded/linux/mtd/mtd_utils/mtd-utils-1.3.1_support_u32u64.7z

第 1 章 嵌入式系统中，如何在Linux运行的时候去升级Linux系统

1.1. 前提

简单点说，在利用mtd工具升级系统之前，需要你的嵌入式linux本身具备一定条件。下面依次介绍这些前提条件。

1.1.1. Linux中已经实现Nor Flash驱动

常见的嵌入式系统，都是从nor flash启动，然后对应的uboot是放在nor flash里面的。

一般nor flash，容量相对较小，只有512KB等，有的大的一点的是1MB，2MB之类的。

一般的情况是，uboot大约有200多KB，而linux的kernel镜像文件，比如我遇到过的，大约在1M左右。

所以，对于这些稍微大一些的Nor Flash，往往除了放了uboot的代码之外，还可以放linux的kernel。

如果是小的Nor Flash，那么往往是把kernel放在Nand Flash的某个分区。

而此处用mtd工具升级linux的前提之一，是你linux系统中，已经实现了对应的nand flash的驱动。而对于nor flash驱动的话，如果还没有实现对应驱动，那么就先去实现对应的nor flash驱动。

下面这里只是对于如何实现普通的nor flash驱动，就我接触到的相关内容，给出一些提示。

对于常见的spi接口的nor flash来说，如果你的nor flash型号是常见的型号，那么很可能你不用另外单独再自己完全从头写一个完整的nor flash驱动了。

关于不同的接口的Nor Flash之间的区别，不了解的可以参考：CFI Flash， JEDEC Flash ，Parellel Flash, SPI Flash， Nand Flash，Nor Flash的区别和联系[1]和CFI（Common Flash Interface）详解[2]

因为，往往你的linux中已经实现了spi驱动的，所以此时，你只需要做下面两件事情，一个是在板子相关部分，添加对应nor flash对应的初始化代码，二是利用linux默认自带的，对于常见nor flash都已经默认支持的nor flash驱动：m25p80.c

下面分别详细解释。

1.1.1.1. 在开发板相关部分添加对应nor flash初始化相关代码

此处，只是简单介绍一下，我之前所遇到的一个nor flash驱动，是如何做的。

关于添加nor flash初始化的代码，其实很简单，就是在开发板的最核心的那个文件（此处以arm系统为例）：

linux-2.6.28.4\arch\arm\mach-XXX\core.c

中，添加类似于这样的代码：

```
static const struct spi_board_info const XXX_spi_devices[] = {
{   /* SSP NOR Flash chip */
 .modalias = "ssp_nor",
 .chip_select = XXX_SPI_NOR_CS,
 .max_speed_hz = 20 * 1000 * 1000,
 .bus_num = 1,
```

[1] http://www.crifan.com/cfi_flash_jedec_flash_parellel_flash_spi_flash_nand_flash_nor_flash_the_differences_and_connections/

[2] http://www.crifan.com/cfi_common_flash_interface_detailed/

```
 },
.....
};
```

然后在自己开发板设备初始化的部分，添加对应spi nor设备的注册函数：

```
spi_register_board_info(XXX_spi_devices,  ARRAY_SIZE(XXX_spi_devices));
```

以实现对应的spi接口的nor flash设备的注册和添加。

具体内部逻辑是如何实现的，就要自己去看代码了。

此处只是给个框架，告诉你大概是怎么去实现的，具体的实现，肯定要你自己去看代码搞懂。

1.1.1.2. Linux通用nor flash驱动m25p80.c简介

在spi接口的nor flash设备注册部分搞定后，再来看Linux中的，默认已经帮我们实现好了的一个通用的nor flash的驱动。

具体的文件是：

linux-2.6.28.4\drivers\mtd\devices\m25p80.c

其中，对于支持的设备，可以去看源码中的设备列表部分的代码：

```
/* NOTE: double check command sets and memory organization when you add
 * more flash chips.  This current list focusses on newer chips, which
 * have been converging on command sets which including JEDEC ID.
 */
static struct flash_info __devinitdata m25p_data [] = {

 /* Atmel -- some are (confusingly) marketed as "DataFlash" */
{ "at25fs010",  0x1f6601, 0, 32 * 1024, 4, SECT_4K, },
{ "at25fs040",  0x1f6604, 0, 64 * 1024, 8, SECT_4K, },

{ "at25df041a", 0x1f4401, 0, 64 * 1024, 8, SECT_4K, },
{ "at25df641",  0x1f4800, 0, 64 * 1024, 128, SECT_4K, },

{ "at26f004",   0x1f0400, 0, 64 * 1024, 8, SECT_4K, },
{ "at26df081a", 0x1f4501, 0, 64 * 1024, 16, SECT_4K, },
{ "at26df161a", 0x1f4601, 0, 64 * 1024, 32, SECT_4K, },
{ "at26df321",  0x1f4701, 0, 64 * 1024, 64, SECT_4K, },

 /* Spansion -- single (large) sector size only, at least
  * for the chips listed here (without boot sectors).
  */
{ "s25sl004a", 0x010212, 0, 64 * 1024, 8, },
{ "s25sl008a", 0x010213, 0, 64 * 1024, 16, },
{ "s25sl016a", 0x010214, 0, 64 * 1024, 32, },
{ "s25sl032a", 0x010215, 0, 64 * 1024, 64, },
{ "s25sl064a", 0x010216, 0, 64 * 1024, 128, },
       { "s25sl12800", 0x012018, 0x0300, 256 * 1024, 64, },
{ "s25sl12801", 0x012018, 0x0301, 64 * 1024, 256, },

/* SST -- large erase sizes are "overlays", "sectors" are 4K */
{ "sst25vf040b", 0xbf258d, 0, 64 * 1024, 8, SECT_4K, },
{ "sst25vf080b", 0xbf258e, 0, 64 * 1024, 16, SECT_4K, },
{ "sst25vf016b", 0xbf2541, 0, 64 * 1024, 32, SECT_4K, },
{ "sst25vf032b", 0xbf254a, 0, 64 * 1024, 64, SECT_4K, },
```

```
/* ST Microelectronics -- newer production may have feature updates */
{ "m25p05",  0x202010,  0, 32 * 1024, 2, },
{ "m25p10",  0x202011,  0, 32 * 1024, 4, },
{ "m25p20",  0x202012,  0, 64 * 1024, 4, },
{ "m25p40",  0x202013,  0, 64 * 1024, 8, },
{ "m25p80",         0,  0, 64 * 1024, 16, },
{ "m25p16",  0x202015,  0, 64 * 1024, 32, },
{ "m25p32",  0x202016,  0, 64 * 1024, 64, },
{ "m25p64",  0x202017,  0, 64 * 1024, 128, },
{ "m25p128", 0x202018, 0, 256 * 1024, 64, },

{ "m45pe80", 0x204014,  0, 64 * 1024, 16, },
{ "m45pe16", 0x204015,  0, 64 * 1024, 32, },

{ "m25pe80", 0x208014,  0, 64 * 1024, 16, },
{ "m25pe16", 0x208015,  0, 64 * 1024, 32, SECT_4K, },

/* Winbond -- w25x "blocks" are 64K, "sectors" are 4KiB */
{ "w25x10", 0xef3011, 0, 64 * 1024, 2, SECT_4K, },
{ "w25x20", 0xef3012, 0, 64 * 1024, 4, SECT_4K, },
{ "w25x40", 0xef3013, 0, 64 * 1024, 8, SECT_4K, },
{ "w25x80", 0xef3014, 0, 64 * 1024, 16, SECT_4K, },
{ "w25x16", 0xef3015, 0, 64 * 1024, 32, SECT_4K, },
{ "w25x32", 0xef3016, 0, 64 * 1024, 64, SECT_4K, },
{ "w25x64", 0xef3017, 0, 64 * 1024, 128, SECT_4K, },
};
```

如果要添加此驱动，以实现支持我们的通用的nor flash，则在make menuconfig的时候，添加对应设备的支持即可。

对应选项的kconfig的配置内容在：

linux-2.6.28.4\drivers\mtd\devices\kconfig

中：

```
config MTD_M25P80
 tristate "Support most SPI Flash chips (AT26DF, M25P, W25X, ...)"
 depends on SPI_MASTER && EXPERIMENTAL
 help
   This enables access to most modern SPI flash chips, used for
   program and data storage.   Series supported include Atmel AT26DF,
   Spansion S25SL, SST 25VF, ST M25P, and Winbond W25X.  Other chips
   are supported as well.  See the driver source for the current list,
   or to add other chips.

   Note that the original DataFlash chips (AT45 series, not AT26DF),
   need an entirely different driver.

   Set up your spi devices with the right board-specific platform data,
   if you want to specify device partitioning or to use a device which
   doesn't support the JEDEC ID instruction.
```

如上所述，如果这些步骤都做完了，最后新编译生成的linux内核，运行后，就应该可以可以通过：

```
cat /proc/mtd
```

查看到对应的mtd设备了。如果没有，那么说明你的驱动还是没有添加正常。

1.1.2. Linux中已实现了U盘挂载，以方便拷贝要升级的文件

简单来说就是，你的linux系统中已经有了USB驱动，并且已经实现了USB的gadget或者USB File storage，即实现了U盘的挂载。

有了U盘挂载，每次升级系统文件，包括uboot，kernel的uImage，rootfs等文件的话，就很方便了。

具体如何实现，不是本文所能说得清楚的，所以不再多赘述。

对于新的Linux内核，在已经实现了USB device驱动的前提下，如何实现U盘的功能，可以参考这个：在Linux USB Gadget下使用U盘[3]

1.1.3. Linux中Nor Flash和Nand Flash已能正常工作

要用mtd工具升级系统之前，肯定是对应的nand flash以及nor flash都是已经正常工作了。即，除了系统正常运行外，通过：

```
cat /proc/mtd
```

可以看到对应的nor和nand的flash所对应的分区信息了。

1.1.4. 已经准备好了mtd工具

此处所说的准备好了mtd的工具，即编译好了某个版本的mtd-utils,比如mtd-utils-1.3.1，然后得到对应的可执行的一系列的工具，其中这几个是用得到的：

表 1.1. MTD工具简介

MTD工具名称	功能简介
flash_erase	擦除（nand或nor）flash的某个部分
flash_eraseall	擦除整个mtd的分区（某个nor或nand分区）
nanddump	用于查看当前某个mtd分区的数据（nand的话,也支持显示oob数据）
nandwrite	用于将某个文件/数据，写入到某个mtd分区（的某个位置）

其中，对于如何得到mtd-util的这些工具，有两种办法：

- 一种是你本身用的buildroot编译的整个rootfs，这时候，可以在配置里面选择上mtd-util的工具，这样生成的出来的rootfs，就有了对应的mtd-util的一系列工具。
- 另一种是，自己去mtd官网下载对应的mtd-util的源码，然后自己编译生成对应的mtd-util的工具。

两种方法，都很简单，只是提醒一下，编译的话，肯定是用交叉编译器，而不是X86的PC上的编辑器去编译，呵呵。

1.1.4.1. mtd-util简介

mtd-util，即mtd的utilities，是mtd相关的很多工具的总称，包括常用的mtdinfo,flash_erase, flash_eraseall, nanddump, nandwrite等，每一个工具，基本上都对应着一个同文件名的C文件。

mtd-util，由mtd官方维护更新，开发这一套工具，目的是为了Linux的MTD层提供一系列工具，方便管理维护mtd分区。

[3] http://www.crifan.com/used_in_the_linux_usb_gadget_u_disk/

mtd工具对应的源码，叫做mtd-utils，随着时间更新，发布了很多版本。

我之前用到的版本是mtd-utils-1.3.1，截止2011-05-01，最新版本到了v1.4.1。

mtd-util源码的下载地址，请去MTD源码的官网[4]

另外多说一句，MTD的官网[5]，资料很丰富，感兴趣的自己去看：

linux的mtd要和mtd-util中的一致

不过，对于之前的版本的Linux的kernel来说，使用mtd-util的话，一定要配套，主要是后来新的linux的版本，开始支持mtd的大小，即nand的大小，大于4GB，对应的linux内核中的mtd层的有些变量，就必须从u32升级成u64，才可以支持。

对应的mtd的util中一些变量，也是要和你当前linux版本的mtd匹配。

简单说就是，无论你用哪个版本的Linux内核，如果要去用mtd-util的话，那么两者的版本要一直，即查看linux内核中的mtd的一些头文件，主要是include\mtd\mtd-abi.h和你的mtd-util中的include\mtd\mtd-abi.h,两个要一致。

否则，就会出现我之前遇到的问题，当然linux内核是u64版本的，支持nand flash大于4GB的，而用的mtd-util中的变量的定义，却还是u32，所以肯定会出错的。

为了同一套mtd-util工具即支持u32又支持u64，我定义了一个宏来切换，下面贴出来，供需要的人参考：

加了宏以支持u32和u64的mtd-abi.h文件

mtd-util中的include\mtd\mtd-abi.h：

```
/*
 * Portions of MTD ABI definition which are shared by kernel and user space
 */

#ifndef __MTD_ABI_H__
#define __MTD_ABI_H__

#include <linux/types.h>

/* from u32 to u64 to support >4GB */
#define U64_VERSION       1

struct erase_info_user {
#if U64_VERSION
 __u64 start;
 __u64 length;
#else
 __u32 start;
 __u32 length;
#endif
};

struct mtd_oob_buf {
#if U64_VERSION
 __u64 start;
#else
```

[4] http://git.infradead.org/mtd-utils.git
[5] http://www.linux-mtd.infradead.org/index.html

```
 __u32 start;
#endif
 __u32 length;
 unsigned char __user *ptr;
};

#define MTD_ABSENT  0
#define MTD_RAM   1
#define MTD_ROM   2
#define MTD_NORFLASH  3
#define MTD_NANDFLASH  4
#define MTD_DATAFLASH  6
#define MTD_UBIVOLUME  7

#define MTD_WRITEABLE  0x400 /* Device is writeable */
#define MTD_BIT_WRITEABLE 0x800 /* Single bits can be flipped */
#define MTD_NO_ERASE  0x1000 /* No erase necessary */
#define MTD_STUPID_LOCK  0x2000 /* Always locked after reset */

// Some common devices / combinations of capabilities
#define MTD_CAP_ROM  0
#define MTD_CAP_RAM  (MTD_WRITEABLE | MTD_BIT_WRITEABLE |
 MTD_NO_ERASE)
#define MTD_CAP_NORFLASH (MTD_WRITEABLE | MTD_BIT_WRITEABLE)
#define MTD_CAP_NANDFLASH (MTD_WRITEABLE)

/* ECC byte placement */
#define MTD_NANDECC_OFF  0 // Switch off ECC (Not recommended)
#define MTD_NANDECC_PLACE 1 // Use the given placement in the structure
 (YAFFS1 legacy mode)
#define MTD_NANDECC_AUTOPLACE 2 // Use the default placement scheme
#define MTD_NANDECC_PLACEONLY 3 // Use the given placement in the structure
 (Do not store ecc result on read)
#define MTD_NANDECC_AUTOPL_USR  4 // Use the given autoplacement scheme
 rather than using the default

#define MTD_MAX_OOBFREE_ENTRIES 8

/* This constant declares the max. oobsize / page, which
 * is supported now. If you add a chip with bigger oobsize/page
 * adjust this accordingly.
 */
#define MTD_NAND_MAX_PAGESIZE 8192
/*
 * for special chip, page/oob is 4K/218,
 * so here alloc more than 256+256 for 8192 pagesize for future special chip like
 that
 */
#define MTD_NAND_MAX_OOBSIZE  (256 + 256)

/* OTP mode selection */
#define MTD_OTP_OFF  0
#define MTD_OTP_FACTORY 1
#define MTD_OTP_USER  2

struct mtd_info_user {
 __u8 type;
 __u32 flags;
```

```
#if U64_VERSION
 __u64 size;  // Total size of the MTD
#else
 __u32 size;  // Total size of the MTD
#endif
 __u32 erasesize;
 __u32 writesize;
 __u32 oobsize;   // Amount of OOB data per block (e.g. 16)
 /* The below two fields are obsolete and broken, do not use them
  * (TODO: remove at some point) */
 __u32 ecctype;
 __u32 eccsize;
};

struct region_info_user {
#if U64_VERSION
 __u64 offset;  /* At which this region starts,
     * from the beginning of the MTD */
#else
 __u32 offset;  /* At which this region starts,
     * from the beginning of the MTD */
#endif
 __u32 erasesize;  /* For this region */
 __u32 numblocks;  /* Number of blocks in this region */
 __u32 regionindex;
};

struct otp_info {
 __u32 start;
 __u32 length;
 __u32 locked;
};

#define MEMGETINFO  _IOR('M', 1, struct mtd_info_user)
#define MEMERASE  _IOW('M', 2, struct erase_info_user)
#define MEMWRITEOOB  _IOWR('M', 3, struct mtd_oob_buf)
#define MEMREADOOB  _IOWR('M', 4, struct mtd_oob_buf)
#define MEMLOCK   _IOW('M', 5, struct erase_info_user)
#define MEMUNLOCK  _IOW('M', 6, struct erase_info_user)
#define MEMGETREGIONCOUNT _IOR('M', 7, int)
#define MEMGETREGIONINFO _IOWR('M', 8, struct region_info_user)
#define MEMSETOOBSEL  _IOW('M', 9, struct nand_oobinfo)
#define MEMGETOOBSEL  _IOR('M', 10, struct nand_oobinfo)
#define MEMGETBADBLOCK  _IOW('M', 11, __kernel_loff_t)
#define MEMSETBADBLOCK  _IOW('M', 12, __kernel_loff_t)
#define OTPSELECT  _IOR('M', 13, int)
#define OTPGETREGIONCOUNT _IOW('M', 14, int)
#define OTPGETREGIONINFO _IOW('M', 15, struct otp_info)
#define OTPLOCK   _IOR('M', 16, struct otp_info)
#define ECCGETLAYOUT  _IOR('M', 17, struct nand_ecclayout)
#define ECCGETSTATS  _IOR('M', 18, struct mtd_ecc_stats)
#define MTDFILEMODE  _IO('M', 19)
/*
 * set/clear prepare oob support
 * usage:
 * 1. set prep_oob_support
 * 2. call write_oob will only prepare, not actually write
 * 3. clear prep_oob_support
```

```
 * 4. write_page will use the previously prepared oob buffer, then clear it
 automatically
 */
#define SETPREPAREOOB  _IOWR('M', 20, int)
#define CLEARPREPAREOOB _IOWR('M', 21, int)

/*
 * Obsolete legacy interface. Keep it in order not to break userspace
 * interfaces
 */
struct nand_oobinfo {
 __u32 useecc;
 __u32 eccbytes;
 __u32 oobfree[MTD_MAX_OOBFREE_ENTRIES][2];
 __u32 eccpos[MTD_NAND_MAX_OOBSIZE];
};

struct nand_oobfree {
 __u32 offset;
 __u32 length;
};

/*
 * ECC layout control structure. Exported to userspace for
 * diagnosis and to allow creation of raw images
 */
struct nand_ecclayout {
 __u32 eccbytes;
 __u32 eccpos[MTD_NAND_MAX_OOBSIZE];
 __u32 oobavail;
 struct nand_oobfree oobfree[MTD_MAX_OOBFREE_ENTRIES];
};

/**
 * struct mtd_ecc_stats - error correction stats
 *
 * @corrected: number of corrected bits
 * @failed: number of uncorrectable errors
 * @badblocks: number of bad blocks in this partition
 * @bbtblocks: number of blocks reserved for bad block tables
 */
struct mtd_ecc_stats {
 __u32 corrected;
 __u32 failed;
 __u32 badblocks;
 __u32 bbtblocks;
};

/*
 * Read/write file modes for access to MTD
 */
enum mtd_file_modes {
 MTD_MODE_NORMAL = MTD_OTP_OFF,
 MTD_MODE_OTP_FACTORY = MTD_OTP_FACTORY,
 MTD_MODE_OTP_USER = MTD_OTP_USER,
 MTD_MODE_RAW,
};
```

```
#endif /* __MTD_ABI_H__ */
```

1.1.4.2. mtd中的/dev/mtdN与/dev/mtdblockN的区别

简单说就是：

- /dev/mtdN

  某个字符设备，对应的mtd的util，就是对其操作，实现对对应的mtd分区进行管理的。

- /dev/mtdblockN

  某个块设备，可以直接像操作其他块设备一样来操作此块设备，比如直接cat数据进去等等常见的操作。

更加详细的解释，请去看这个帖子：

Linux系统中/dev/mtd与/dev/mtdblock的区别，即MTD字符设备和块设备的区别 [6]

1.2. 准备工作

1.2.1. 准备好要升级的文件

将你新编译和制作出来的，要升级的文件准备好，此处为：

表 1.2. 要升级的Linux系统的文件

文件	文件名	说明
uboot文件	u-boot.bin	只是一个普通的二进制文件
linux的kernel文件	uImage	也是一个普通的二进制文件
rootfs文件	rootfs.4k.arm.yaffs2	是用mkyafffs2工具制作而成，内部数据格式是page数据+oob数据+page数据+oob数据+.....，用于烧写到Nand Flash中

1.2.2. 拷贝文件并挂载分区

此处，我的系统的U盘，是挂载在/dev/mtdblock4中。

所以，先要通过挂载/dev/mtdblock4，即Data分区,作为U盘到PC上，

拷贝要升级的文件和util文件夹及其下面的工具：

nandwrite,flash_erase,flash_eraseall,nanddump

到U盘上，然后弹出U盘，之后将/dev/mtdblock4挂载到/mnt/dos下

此时，/mnt/dos下就该有

- u-boot.bin
- uImage
- rootfs.4k.arm.yaffs2
- util/

[6] http://www.crifan.com/linux_system_in__dev__mtd_and__dev__mtdblock_distinction_character_devices_and_block_devices_mtd_difference/

1.3. 利用mtd工具升级Linux系统

利用mtd工具升级系统，其实说白了，就是：

1. 用flasherase擦除数据
   先用flasherase擦除对应mtd分区中的内容

2. 用nandwrite写入数据
   然后将对应的数据（uboot或uImage或rootfs）用nandwrite写入到对应的mtd中对应的位置即可。

前面介绍过了，对于常见的是把uboot（和kernel）放到nor flash中，而把kernel和rootfs放在nand flash中的。

而我此处的举的例子，是另外一种，即全部内容都放在nand flash上的。

但是，不论是是nor flash，还是nand flash，都在Linux的MTD框架下，管理起来，都是一样的。都是可以用对应的mtd的工具去操作的。所以，如果你本身是要升级对应的uboot（和kernel）到nor flash，对于整个过程，也是一样的，自己照葫芦画瓢即可。

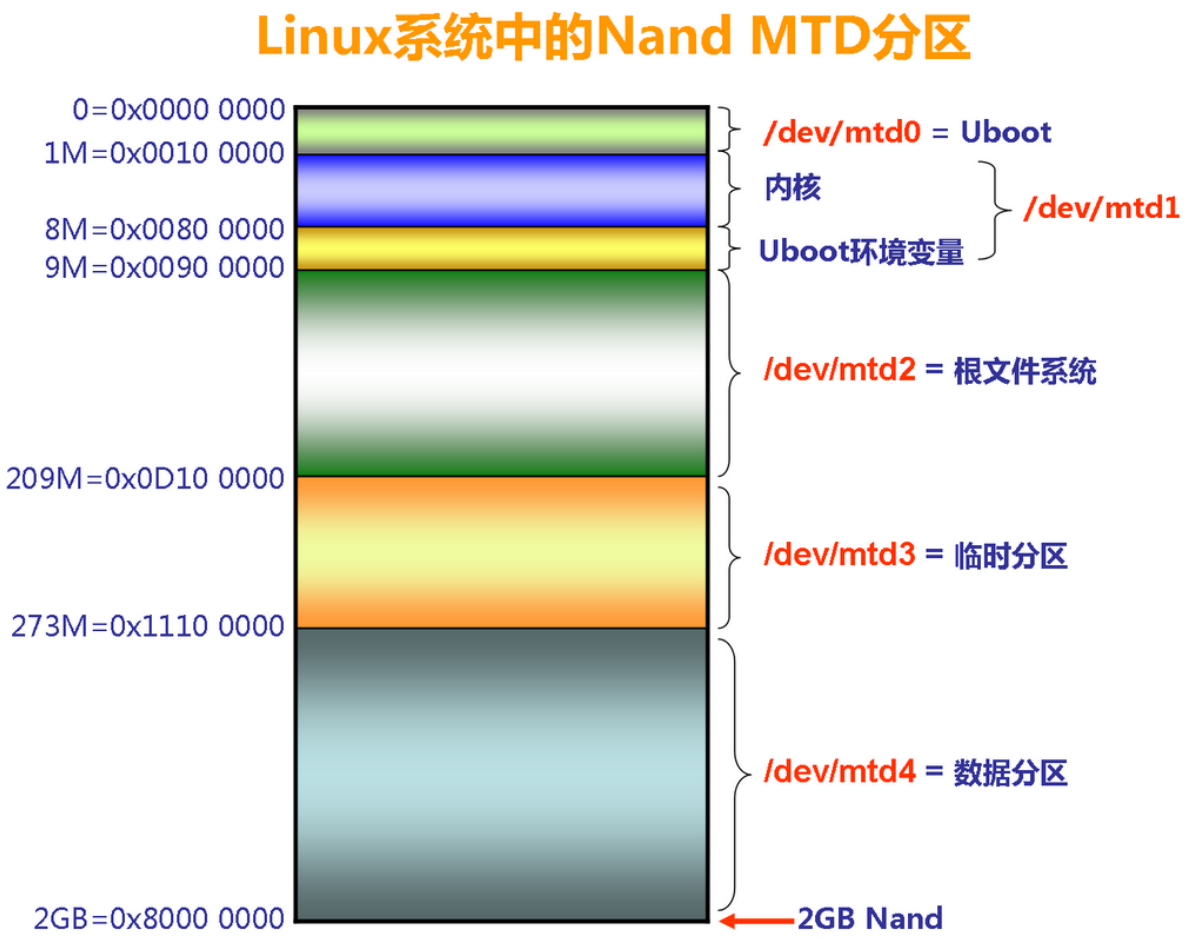
关于我此处举例所用的MTD的分区是如何的，此处先给出相关部分的代码：

```
#define UBOOT_SIZE  (SZ_1M)
#define KERNEL_SIZE  (SZ_8M)
#define ROOTFS_SIZE (SZ_1M*200)
#define TEMP_SIZE  (SZ_1M*64)

#define BEFORE_DATA_PARTION_SIZE      \
  (ROOTFS_SIZE + KERNEL_SIZE + UBOOT_SIZE + TEMP_SIZE)
。。。
static struct mtd_partition XXX_default_nand_part[] = {
 [0] = {
  .name = "U-Boot",
  .offset = 0,
  .size  = UBOOT_SIZE,
 },
 [1] = {
  .name = "Kernel",
  .offset  = UBOOT_SIZE,
  .size  = KERNEL_SIZE
 },
 [2] = {
  .name = "Root filesystem",
  .offset  = UBOOT_SIZE + KERNEL_SIZE,
  .size  = ROOTFS_SIZE,
 },
 [3] = {
  .name = "Temp",
  .offset  = UBOOT_SIZE + KERNEL_SIZE + ROOTFS_SIZE,
  .size  = TEMP_SIZE,
 },
 [4] = {
  .name = "Data",
  .offset = BEFORE_DATA_PARTION_SIZE,
  .size  = 0, /* set in XXX_init_nand_partion() */
 },
};
```

对应的用图表来说明，如下：

图 1.1. Linux系统中的Nand MTD分区



下面就来介绍，如何一步步升级uboot，kernel和rootfs。

1.3.1. 升级Uboot

1. 擦除uboot所在分区的所有数据

```
./util/flash_eraseall /dev/mtd0
```

2. 擦除旧的uboot的环境变量

```
./util/flash_erase /dev/mtd1 0x700000 2
```

- 0x800000~0x900000
  即/dev/mtd1中的0x700000~0x800000, 用于存放uboot中的环境变量。

  重新升级uboot的同时，先把旧的环境变量擦除掉。

3. 写入uboot数据

```
./util/nandwrite -p -s 0x80000 /dev/mtd0 u-boot_addHeader.bin
```

- -p参数

表示如果要写入的数据不是页大小的整数倍，会自己加填充数据即，如需要，自动padding。

- 0x80000
  是当前4K的pagesize的nand flash的一个块的大小。

1.3.2. 升级Kernel

1. 擦除旧的kernel数据

```
./util/flash_erase /dev/mtd1 0 10
```

- 其中的参数0，表示从/dev/mtd1起始位置开始擦除
- 参数10是表示要擦除的block数目
  /dev/mtd1的物理起始地址是0x100000，而0x100000~0x600000之间，是用于保存uImage的数据，所以：

  要擦除的block的数目

  = 要擦除的大小/块大小

  = 0x500000/块大小

  = 5M/512KB

  = 10

  其中，当前用的是这个4K pagesize的nand的块大小是512KB。

2. 写入kernel数据

```
./util/nandwrite -p /dev/mtd1 uImage
```

1.3.3. 升级rootfs

1. 擦除rootfs所在分区数据

```
./util/flash_eraseall /dev/mtd2
```

2. 写入新的rootfs

```
./util/nandwrite -o /dev/mtd2 rootfs.4k.arm.yaffs2
```

- 因为此处的rootfs镜像文件是yaffs2文件系统，包含了oob数据。所以此处加上参数-o，意思是写入页数据同时也写入oob数据，而且，加了-o 参数同时就不能再像之前的uboot和uImage一样，加-p参数了，因为包含了oob数据的rootfs，本身就是页大小的整数倍，不需要padding。
- 不论实际使用的是4K+128 还是对于4K+218（内部处理为4K+192）的nand，此处都是使用4K+128的rootfs镜像。

1.4. 总结整个升级过程

整个runtime的升级linux的过程，其实很简单。

如果说有难度的话，那么算是，在升级数据之前，你自己本身要清楚你原先的数据，即
uboot，kernel，rootfs，都是放在哪个分区的哪个位置的，然后分别擦除数据，写入新数据即可。

另外有个要注意的是，升级rootfs的话，尽量把其他非内核必须的进程都关闭掉，防止在升级过程中，还有进程或和程序去读取nand flash上的rootfs。

此外，在烧写某个文件之后，如果希望查看当前写入的数据，是否是我们所期望的，那么可以用nanddump工具，将对应部分的数据“打印”出来，比如：

查看uboot的第一page的数据：

```
./nanddump -l 0x1000 -s 0x80000 -p /dev/mtd0
```

其他mtd-util的工具的用法，请自己参考mtd-util中源码的具体实现，通过看源码，可以了解其具体是如何实现，以及参数的完整的含义。

1.4.1. 一些提示

1.4.1.1. 把东西放到ramdisk中以避免影响

之前遇到很多人问这个问题了。那就是，如果在升级的时候，由于也会升级rootfs，但是本身升级过程中，所利用到的文件，如果是放在rootfs中，岂不是会导致系统崩溃了？

答案是，不会。因为我之前介绍的方法中，是把升级所需的mtd工具，放到U盘的。而U盘是单独mount系统中的。

不过，更加好的做法是，把此处升级所相关的，所有的文件，包括mtd工具，要升级的各个文件，甚至其他可能用到的reboot等工具，设置是这些工具可能依赖的到库文件等等，都全部拷贝到ramdisk中。这样，通过运行ramdisk中的所有工具，访问ramdisk中的要升级的文件，去升级系统，就不会对升级rootfs而有啥负面影响，也不会由于升级rootfs而可能导致任何的系统崩溃了。

关于ramdisk

所谓的ramdisk，我个人也不是非常熟悉。

只是对更不熟悉的人解释一下，可以简单理解为把你的内存划分出来，当做一块分区使用

所以这个小分区，说白了就是内存。所以，读写速度很快，也和nand 或nor flash无关，不会影响到Nand或Nor的升级。

一般来说，多数都是将ramdisk挂载到/tmp下面的，所以，如果你啥都不熟悉，直接把相关文件拷贝到/tmp，即可。

更多的，关于ramdisk或tmpfs，自己google吧。